Fountain: The Node Monitoring Component of a Scalable Systems Software Environment

Master’s Oral Thesis Defense
major: Computer Engineering

Sam Miller
Scalable Computing Laboratory
Iowa State University
July 7, 2006

committee members

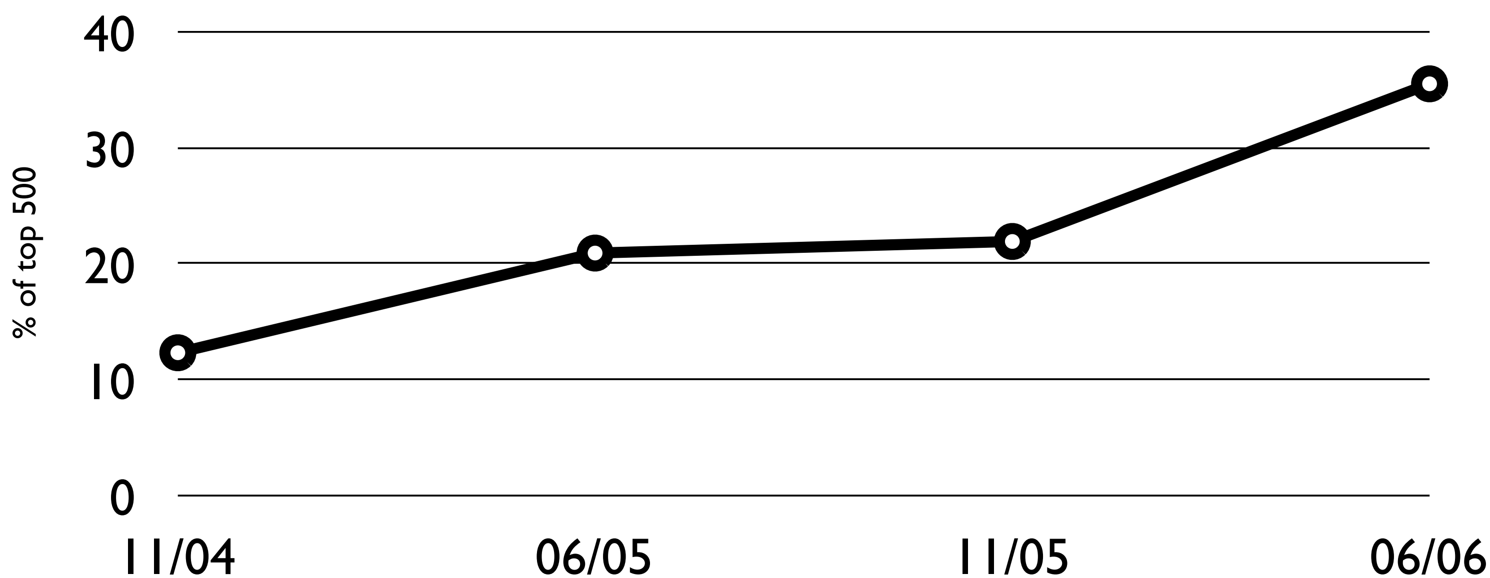
Brett Bode	Srinivas Aluru	Robyn Lutz
ECpE	ECpE	CS

Outline

- Problem statement & motivation (9 slides)
- Prior works (5 slides)
- System design (21 slides)
- Extensibility (9 slides)
- Results (12 slides)
- Conclusion (2 slides)

Problem Statement 1/4

- Parallel system size is expanding
- Percentage of top 500 systems with greater than 1,025 processors:



Problem Statement 2/4

- Scalability on hardware end is great
- Cluster management software does not scale as well as the hardware does
- Computer industry not motivated to solve this problem
- Home-grown management software will need to be re-done as larger clusters are installed

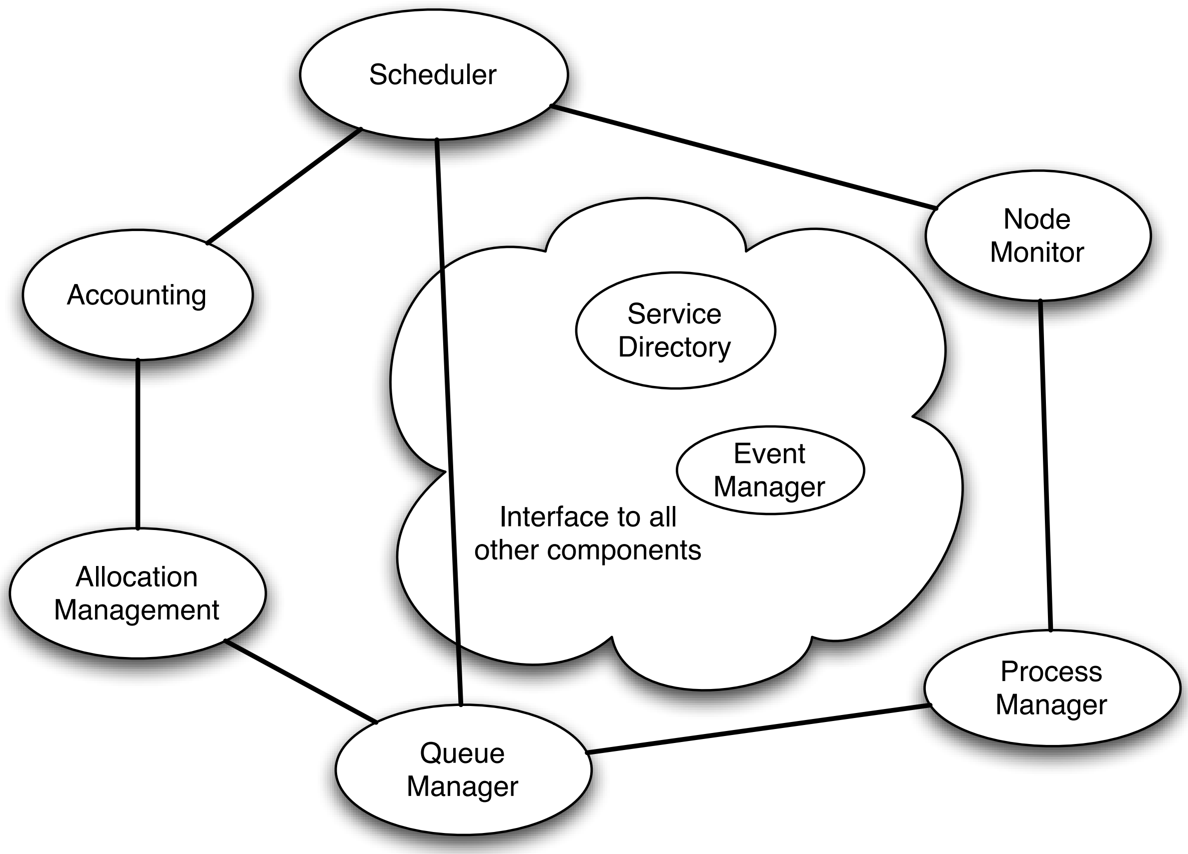
Problem Statement 3/4

- Many resource management systems exist
- Portable Batch System (PBS) is popular
  - Provide services to run user jobs, monitor cluster status, collect output, etc.
  - Transient connections are inefficient and not scalable
  - Larger clusters require more effective solutions

Problem Statement 4/4

- Scalable Systems Software (SSS) effort
  - Part of DOE SciDAC program
  - Desire is to more effectively utilize next generation computational resources
  - Goal is to develop component based open source cluster management software
  - Modularity is a key goal, allows for specialized components

Scalable Systems Software



Node Monitoring 1/4

- What is monitoring?
  - Act of observing a signal
  - Either reactive or periodic in nature
- What is node monitoring?
  - Observe hard drive status, CPU usage, fan speed, power supply temp, etc.
  - collect this information for all cluster nodes
  - node state is important

Node Monitoring 2/4

- Why is node monitoring necessary?
  - Large clusters utilize batch systems to schedule and run user jobs
  - Effective scheduling requires accurate node status
  - Presents a single system image to a system administrator

Node Monitoring 3/4

- Our node monitor is called Fountain
- Three distinct design goals
  - Fault tolerant to handle node failures
  - Low processing requirements
  - Scalable to next generation hardware
- What was our motivation?
  - Provide cluster scheduler with node state

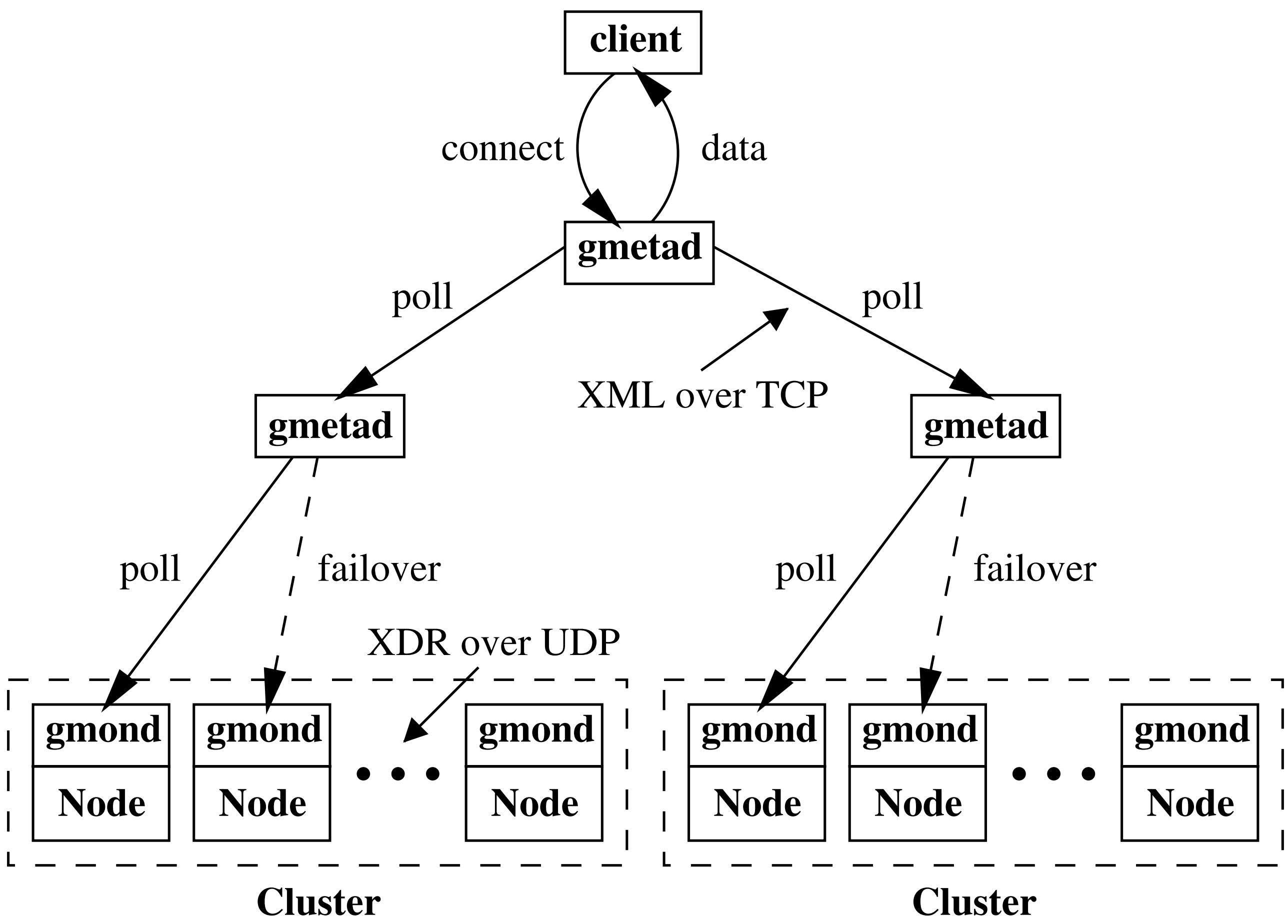
Node Monitoring 4/4

- Prior works
  - Ganglia - University of California Berkeley
  - Supermon - Los Alamos National Lab
  - NWPerf - Pacific Northwest National Lab

Ganglia 1/2

- Designed for clusters and grids
- Relies on multicast listen/announce protocol
- Configurable publishing thresholds for different monitoring metrics
- Three components: clients, gmetad, gmond
- Novelty: minimal configuration
- Drawback: no cluster scheduler interface

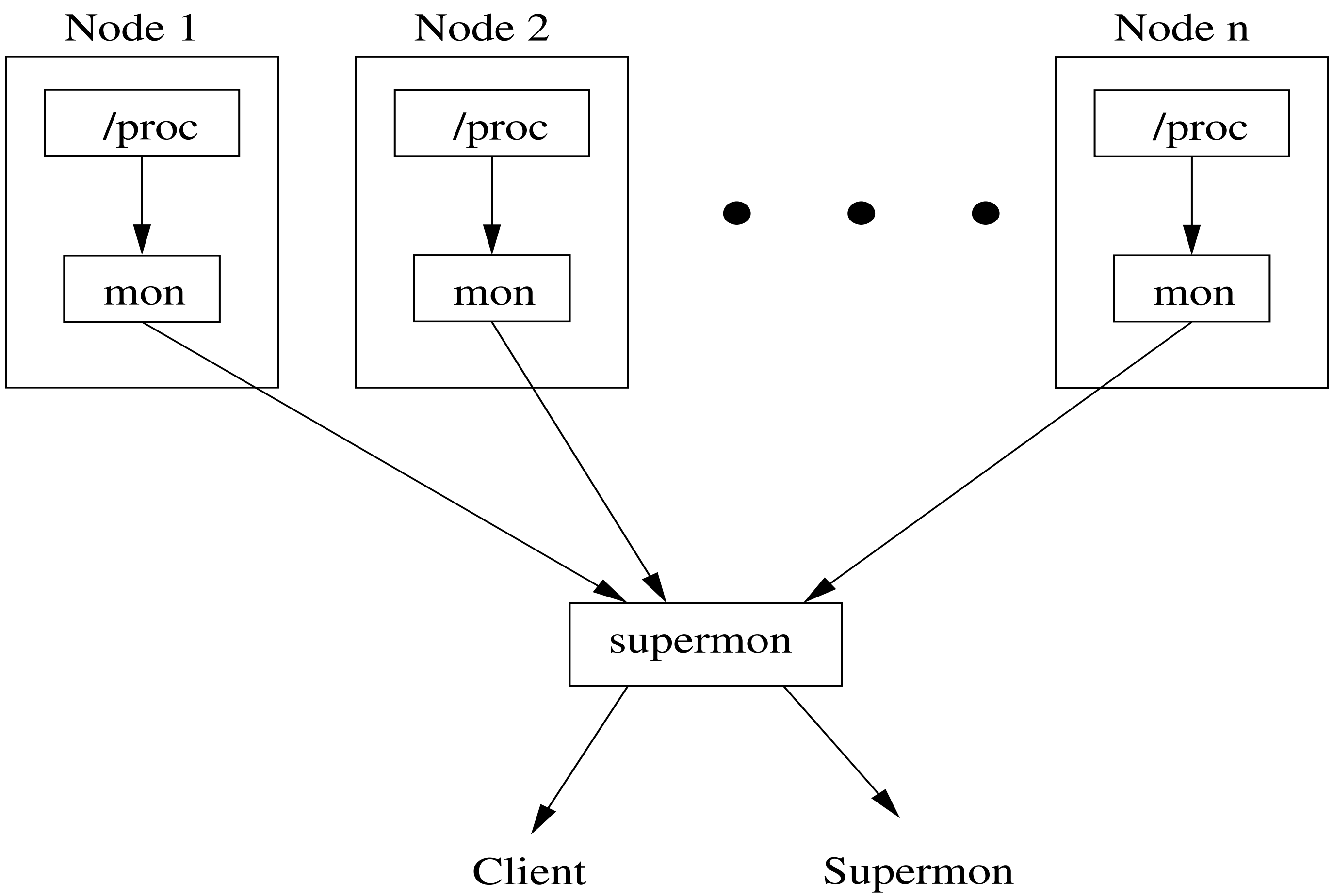
Ganglia 2/2


client
connect
data
gmetad
poll
poll
XML over TCP
gmetad
gmetad
poll
failover
poll
failover
XDR over UDP
gmond
Node
gmond
Node
gmond
Node
gmond
Node
gmond
Node
gmond
Node
Cluster
Cluster

Supermon 1/2

- Designed for high speed monitoring
- Three components: kernel module, node daemon, data aggregator
- Each uses symbolic expressions (S-expressions from LISP)
- Novelty: very high speed
- Drawback: no memory usage monitoring, no cluster scheduler interface

Supermon 2/2


Node 1
Node 2
Node n
/proc
mon
/proc
mon
/proc
mon
supermon
Client
Supermon

NWPerf

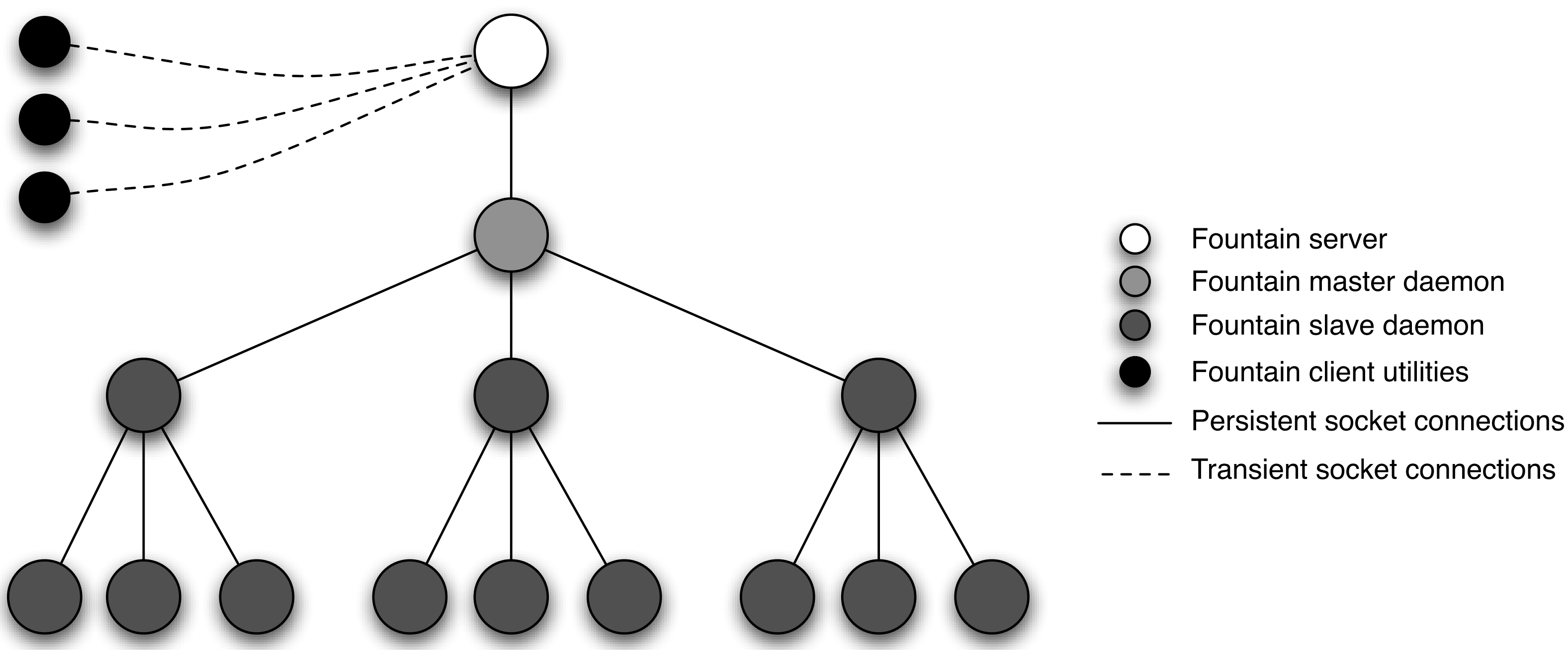
- Designed for low-impact, high resolution monitoring
- Goal is to monitor behavior of user applications
- Three components: lightweight client per node, server side packet handler, external storage system
- Drawback: no scheduler interface, targeted more towards analysis purposes

Fountain Design 1/3

- Four separate components
  - server (head node)
  - master daemon (head node)
  - slave daemon (each compute node)
  - client utilities (anywhere)

Fountain Design 2/3

overview of the four Fountain components and
how they interact with each other



Fountain Design 3/3

- Components communicate using XML messages over TCP sockets
- Both persistent and transient sockets
- Required environment:
    - Linux
    - TCP connected hosts

Slave daemon

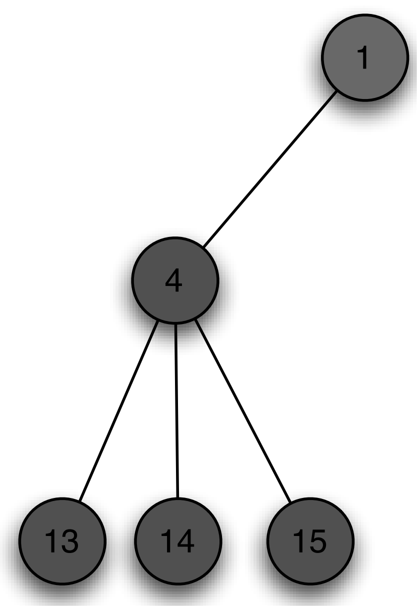
- Slave Fountain daemons run on each compute node in the cluster, how they start is not important
- Arranged in a rigid topology by the master daemon (more info later)
- Two purposes
  - collect monitored metrics
  - report neighboring daemon failures

Monitored Metrics

- Static: CPU, memory, swap space
- Dynamic: CPU usage, available memory, available swap
- Collected from the /proc file system in Linux
- node state is NOT collected, more on this later

Slave daemon

- Each slave daemon has:
  - a persistent connection to a parent daemon
  - up to n persistent connections to child daemons
  - number of children depend on tree topology config



Slave daemon

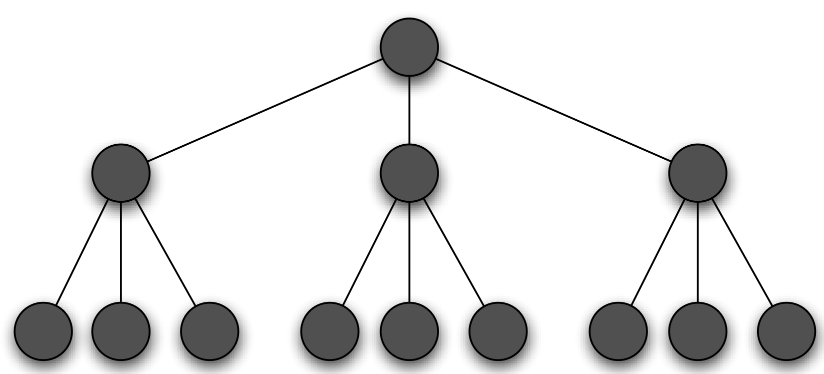
- Monitoring metrics are collected on demand by Fountain server (more on this later)
- Otherwise slave daemons are essentially idle
  - sleep in a select system call waiting for I/O
  - Promotes low node overhead

Master daemon

- Same functionality as slave daemon
- Added requirement of maintaining topology of slave daemons
  - We chose a tree topology
  - Promotes good scalability
  - Recovering from node failures is somewhat difficult

Master daemon

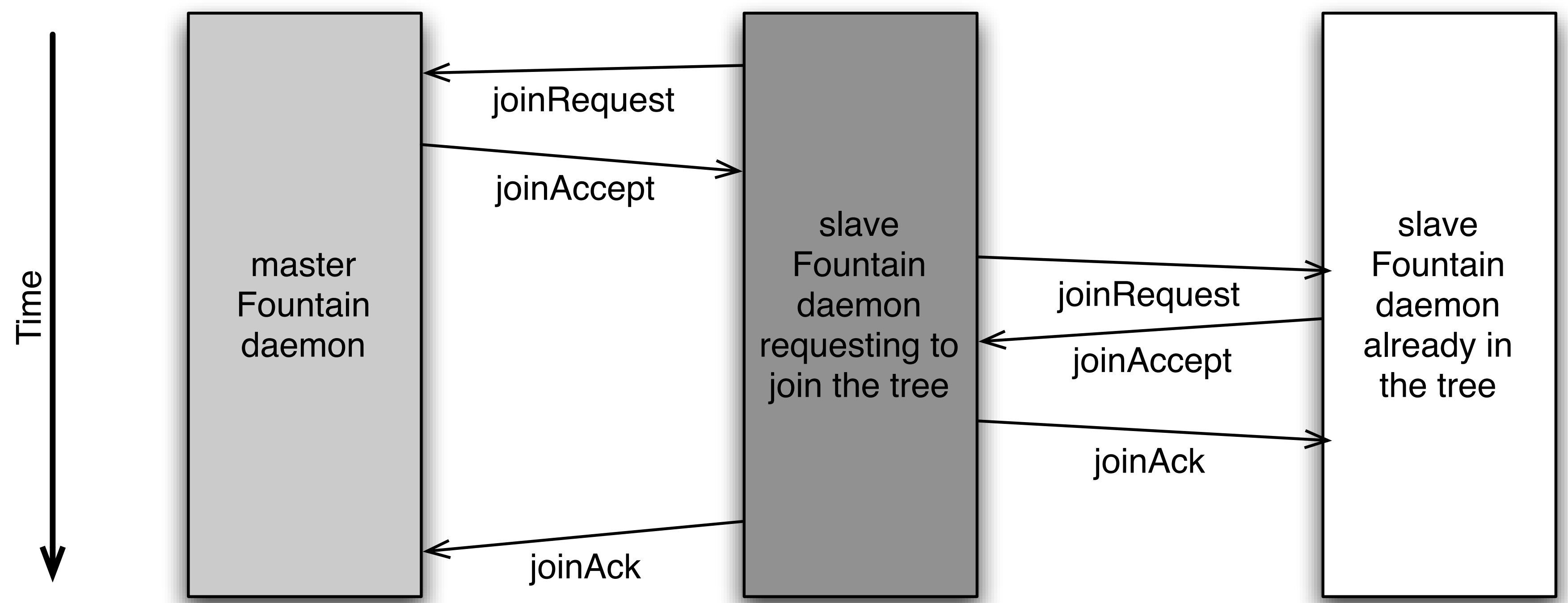
- Fountain tree topology is a complete n-ary topology
  - Each node has up to n children
  - Each level is full except the bottom level
  - Bottom level is filled left to right



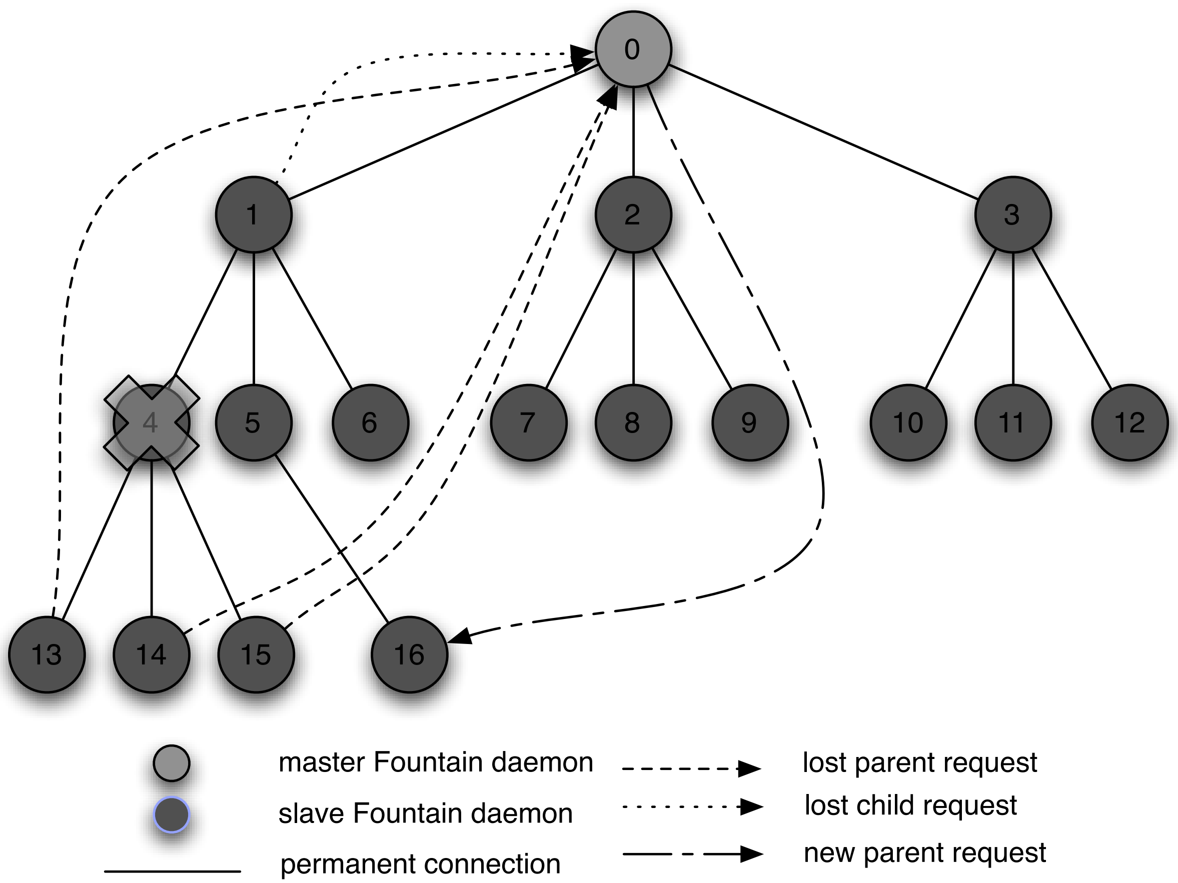
Master daemon

- Fountain uses three algorithms to maintain the tree topology in the presence of failures
  - Tree establishment
  - Tree recovery
  - Tree rebuilding

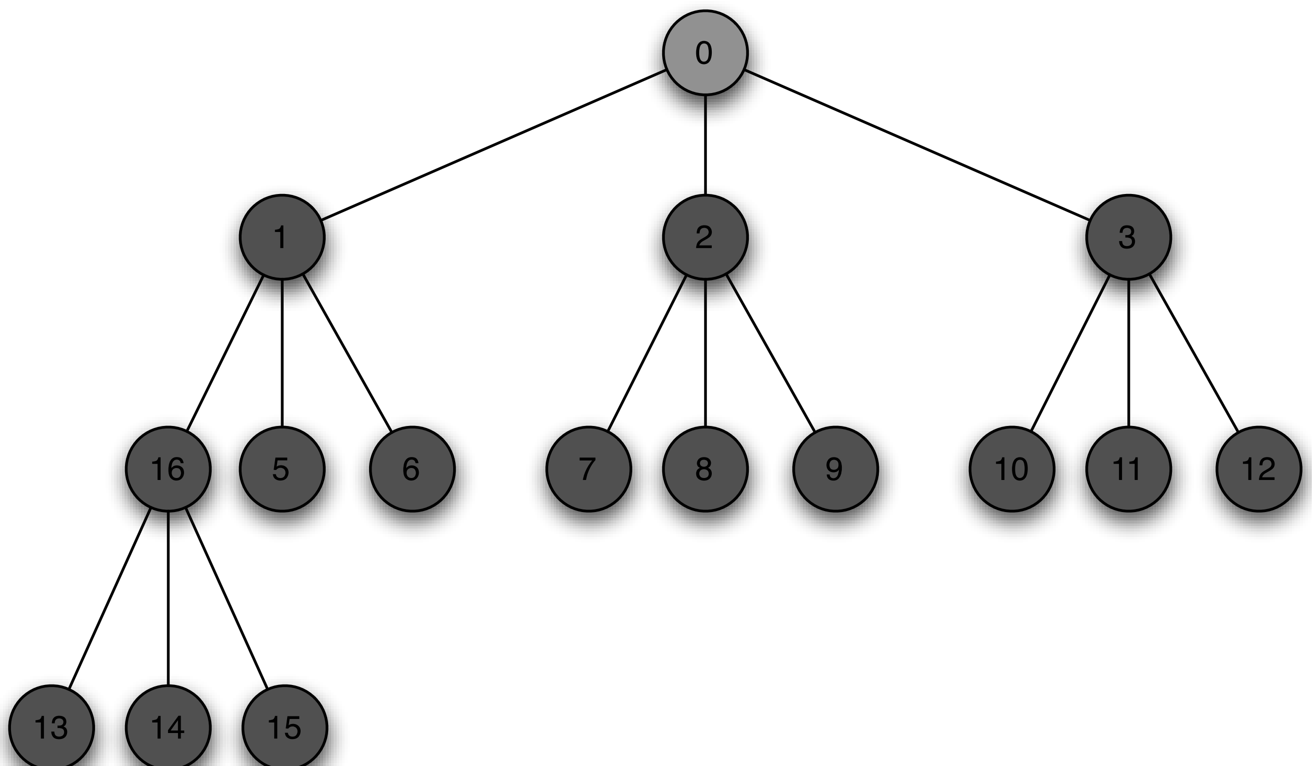
Tree Establishment



Tree Recovery 1/3

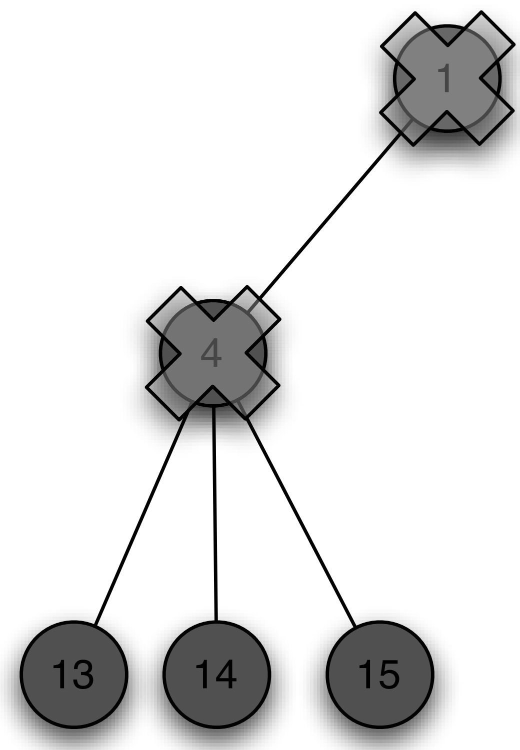


Tree Recovery 2/3

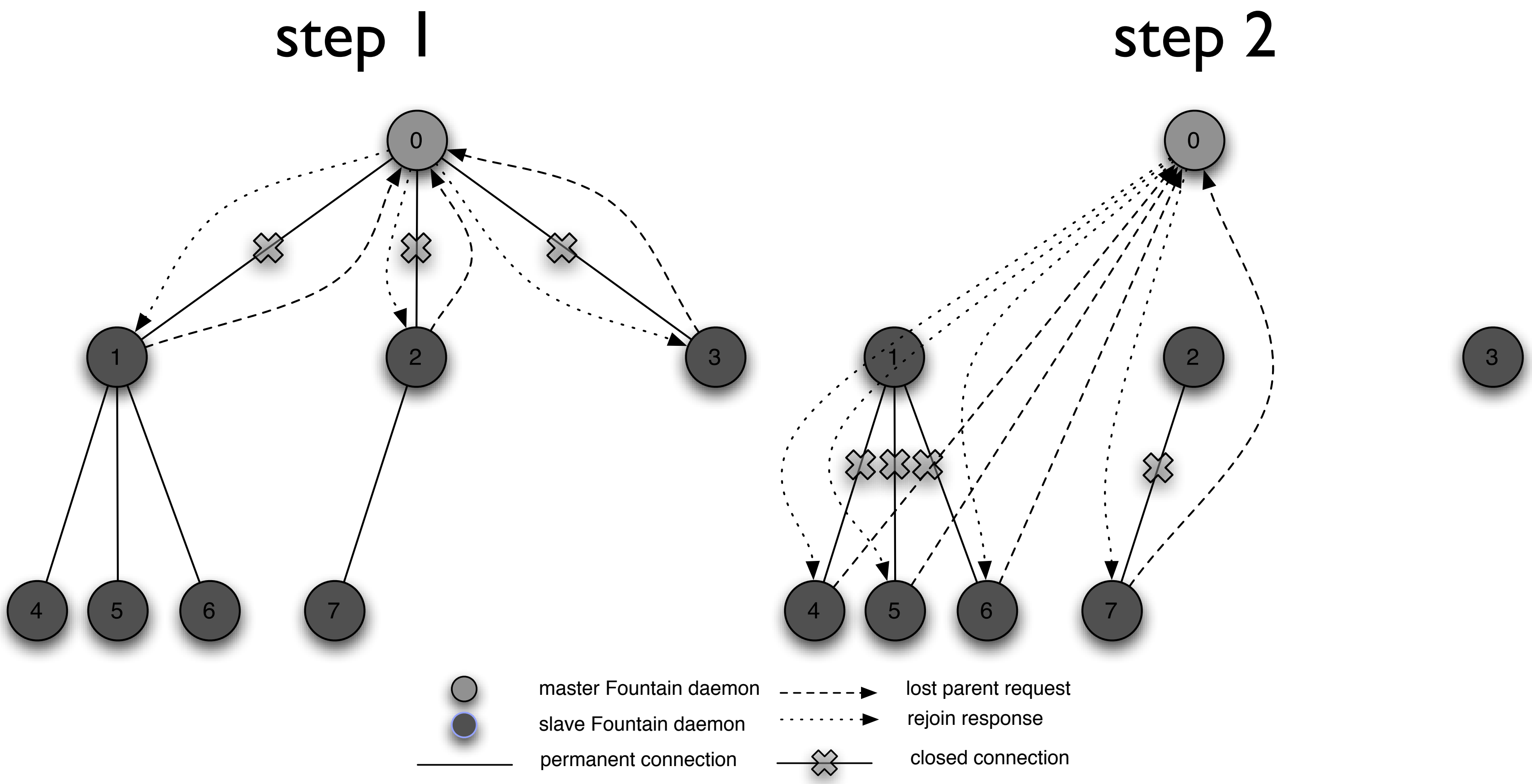


Tree Recovery 3/3

- Why wait for all neighbors to report failure?
  - assume only the parent reports failure
  - what happens to nodes 13, 14, and 15 when nodes 1 and 4 fail concurrently?



Tree Rebuilding



Fountain server 1/5

- Acts as a gateway between Fountain daemons and other SSS components
- Presents a single system image to clients
    - stores monitoring info from daemons
    - Responds to client requests
- It has a very flexible interface by utilizing the SSS Node Monitor and Node Object spec

Fountain server 2/5

Request:

```
<Envelope>
  <Body actor="samm">
    <Request action="Query">
      <Object>Node</Object>
      <Get name="NodeId"></Get>
      <Get name="NodeState"></Get>
      <Where name="NodeState" op="eq">Down</Where>
    </Request>
  </Body>
</Envelope>
```

Response:

```
<Envelope>
  <Body actor="root">
    <Response action="Query">
      <Count>2</Count>
      <Total>34</Total>
      <Data name="NodeList" type="xml">
        <Node>
          <NodeId>m20</NodeId>
          <State>Down</State>
        </Node>
        <Node>
          <NodeId>m34</NodeId>
          <State>Down</State>
        </Node>
      </Data>
      <Status>
        <Value>Success</Value>
        <Code>000</Code>
        <Message>2 node(s) found</Message>
      </Status>
    </Response>
  </Body>
</Envelope>
```

Fountain server 3/5

Request:


```
<Envelope>
  <Body actor="root">
    <Request action="Query">
      <Object>Node</Object>
      <Get name="NodeId"></Get>
      <Get name="Arch"></Get>
      <Get name="OpSys"></Get>
      <Get name="State"></Get>
      <Where name="State" op="eq">Up</Where>
      <Get name="Configured/Processors"></Get>
      <Where name="Configured/Processors" op="ge">2</Where>
      <Get name="Available/Memory" units="MB"></Get>
      <Where name="Available/Memory" op="ge" units="MB">128</Where>
    </Request>
  </Body>
</Envelope>
```


Response:


```
<Envelope>
  <Body actor="root">
    <Response action="Query">
      <Count>1</Count>
      <Total>34</Total>
      <Data name="NodeList" type="xml">
        <Node>
          <State>Up</State>
          <NodeId>m17</NodeId>
          <Arch>ppc</Arch>
          <OpSys>Linux</OpSys>
          <Configured>
            <Processors>2</Processors>
          </Configured>
          <Available>
            <Memory units="MB">819.5</Memory>
          </Available>
        </Node>
      </Data>
      <Status>
        <Value>Success</Value>
        <Code>000</Code>
        <Message>1 node(s) found</Message>
      </Status>
    </Response>
  </Body>
</Envelope>
```

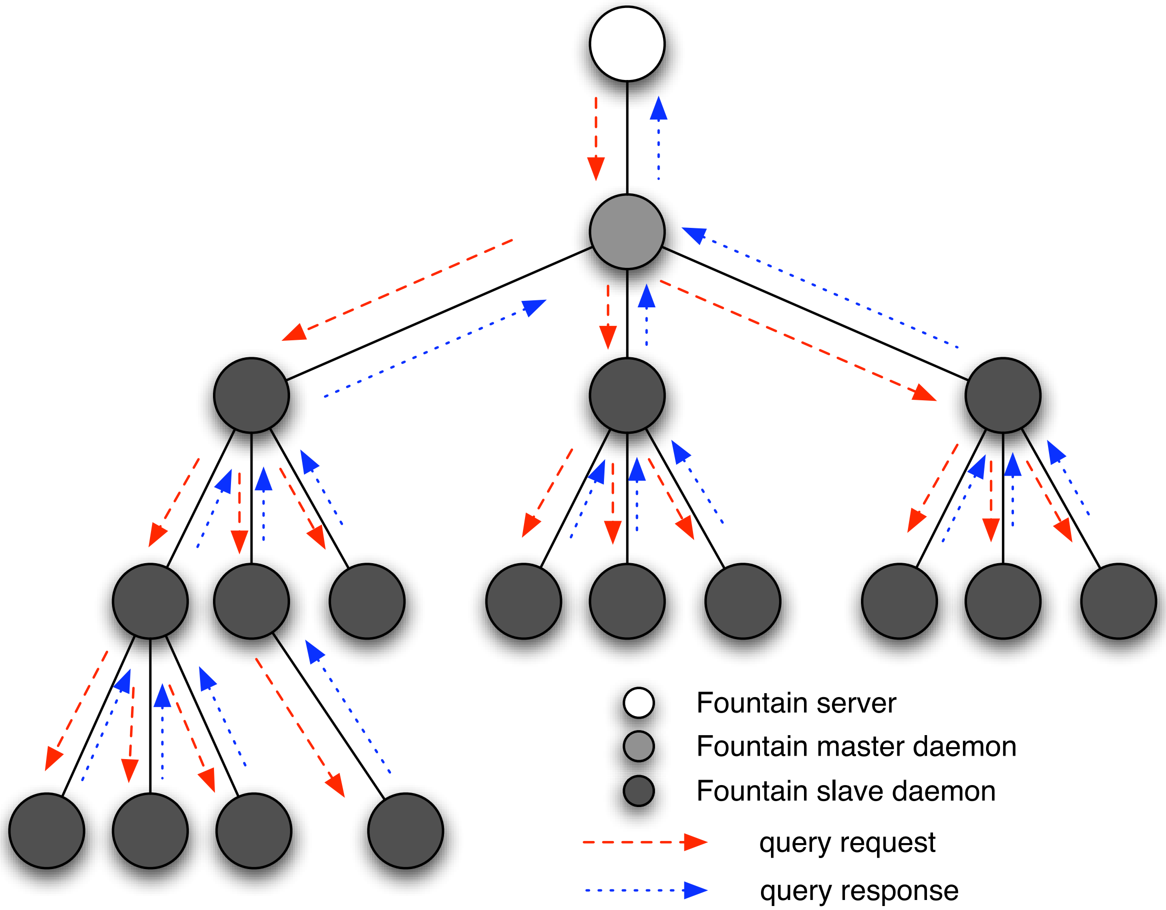
Fountain server 4/5

- Utilizes a node monitor database
  - Actually a C++ map container from STL
  - Node's hostname is the key, object to hold node statistics is the value
- Three ways to populate this database
  - Query response from Fountain daemons
  - Parsing nodelist file
  - Discovering a server-specific data source

Node Query 1/2

- Fountain server periodically sends a query request to the master daemon
  - master responds w/ query response
  - message contains info for all daemons
- Node state calculated from query response
  - Three states: up, down, unavailable

Node Query 2/2



Extensibility 1/2

- Thus far, only node specific data sources have been discussed
- Other potential sources for monitoring information exist
  - Parallel file systems (PVFS, GPFS, Lustre)
  - Network information (gigabit ethernet, InfiniBand, Myrinet)
- Such information could be beneficial

Extensibility 2/2

- Fountain currently has two modules to extend its monitoring capabilities
  - InfiniBand module
  - Cray XT3 module
- Integrated into the Fountain server
- In some cases, node daemon functionality is disabled because it does not make sense

InfiniBand 1/5

- Modern interconnection architecture
  - 3rd most popular on 27th top 500 list
  - behind Myrinet (2nd) and gigabit ethernet (1st)
- Utilizes a bidirectional serial bus
- Links can be aggregated: 1x, 4x, 12x
- 12X double data rate (DDR) can carry 60 gigabits/second

InfiniBand 2/5

- Open InfiniBand Alliance (OpenIB)
  - Open source IB software stack
  - Supports HCAs from multiple vendors
  - Interface accepted into Linux kernel
- We desired two features
  - Discover IB network
  - Poll each discovered node for port counter information

InfiniBand 3/5

- Motivation for this module came from SC|05
  - Desire was to make a visual map of nodes
  - Overlay performance & error counters
- Extend this idea to clusters
  - User can overlay network topology to monitor job status

InfiniBand 4/5

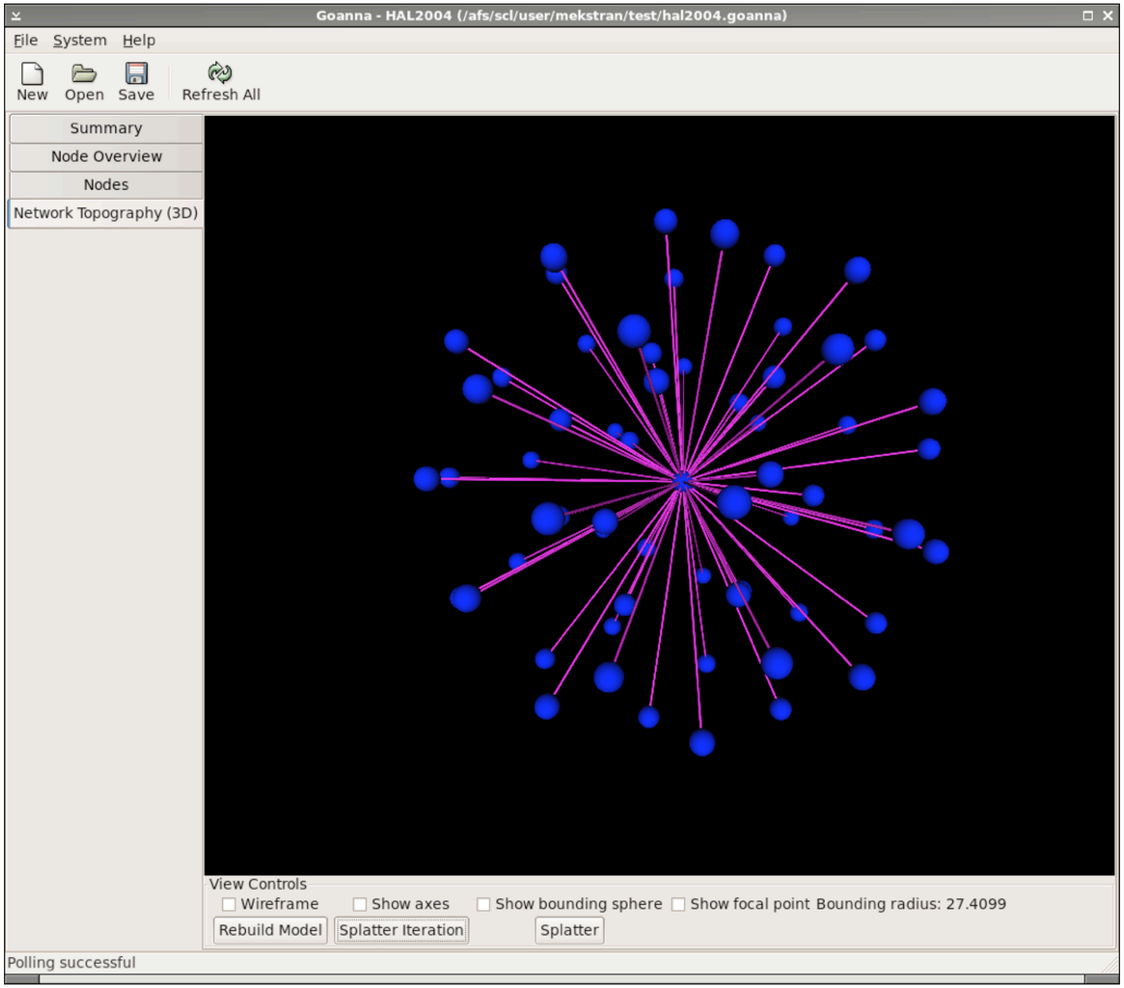
Request:


```
<Envelope>
  <Body actor="samm">
    <Request action="Query">
      <Object>Node</Object>
      <Get name="NodeId"></Get>
      <Get name="Network"></Get>
    </Request>
  </Body>
</Envelope>
```


Response:


```
<Node>
  <NodeId>0002c90200003448</NodeId>
  <Arch>Infiniband</Arch>
  <Network type="Infiniband">
  <Device>
    <ID>0002c90200003448</ID>
    <Vendor>Redswitch</Vendor>
    <Lid>35</Lid>
    <Description>MT23108 InfiniHost Mellanox Technologies</Description>
    <Type>HCA</Type>
    <Ports>
    <PortCount>2</PortCount>
    <Port>
      <Number>1</Number>
      <RemoteDevice port="2">0002c90109fb36b8</RemoteDevice>
      <SendBytes units="bytes">648</SendBytes>
      <ReceiveBytes units="bytes">576</ReceiveBytes>
      <SendRate>
        <Bytes>39.865</Bytes>
        <Packets>0.554</Packets>
      </SendRate>
      <ReceiveRate>
        <Bytes>39.865</Bytes>
        <Packets>0.554</Packets>
      </ReceiveRate>
      <SymbolErrors>60</SymbolErrors>
      <Counters>true</Counters>
      <LastSeen>Mon Jun 5 15:09:38 2006</LastSeen>
      <Width>4X</Width>
      <Speed units="Gigabits/sec">2.5</Speed>
    </Port>
    </Ports>
  </Device>
  </Network>
</Node>
```

InfiniBand 5/5



Cray XT3 1/2

- Massively parallel processing (MPP) system
- Developed by Sandia and Cray Inc.
- Contains between 200 and up to 30,000 processors
- Built-in management software presents a single system image

Cray XT3 2/2

- Fountain module for XT3 acts as a wrapper
  - Discovers number of installed processors
  - Updates number of available processors periodically
  - Provides this information to the cluster scheduler
- Not yet feature complete
- Created to test feasibility of this application

Test Environment 1/3

- Two test environments
  - Scink: 64 node dual AMD Athlon MP2200 cluster with 100 Mbit ethernet
  - 4pack: 34 node heterogeneous PowerPC G4 Macintosh cluster
  - Both run Debian Linux
  - Larger configurations are tested with multiple Fountain daemons per node

Test Environment 2/3

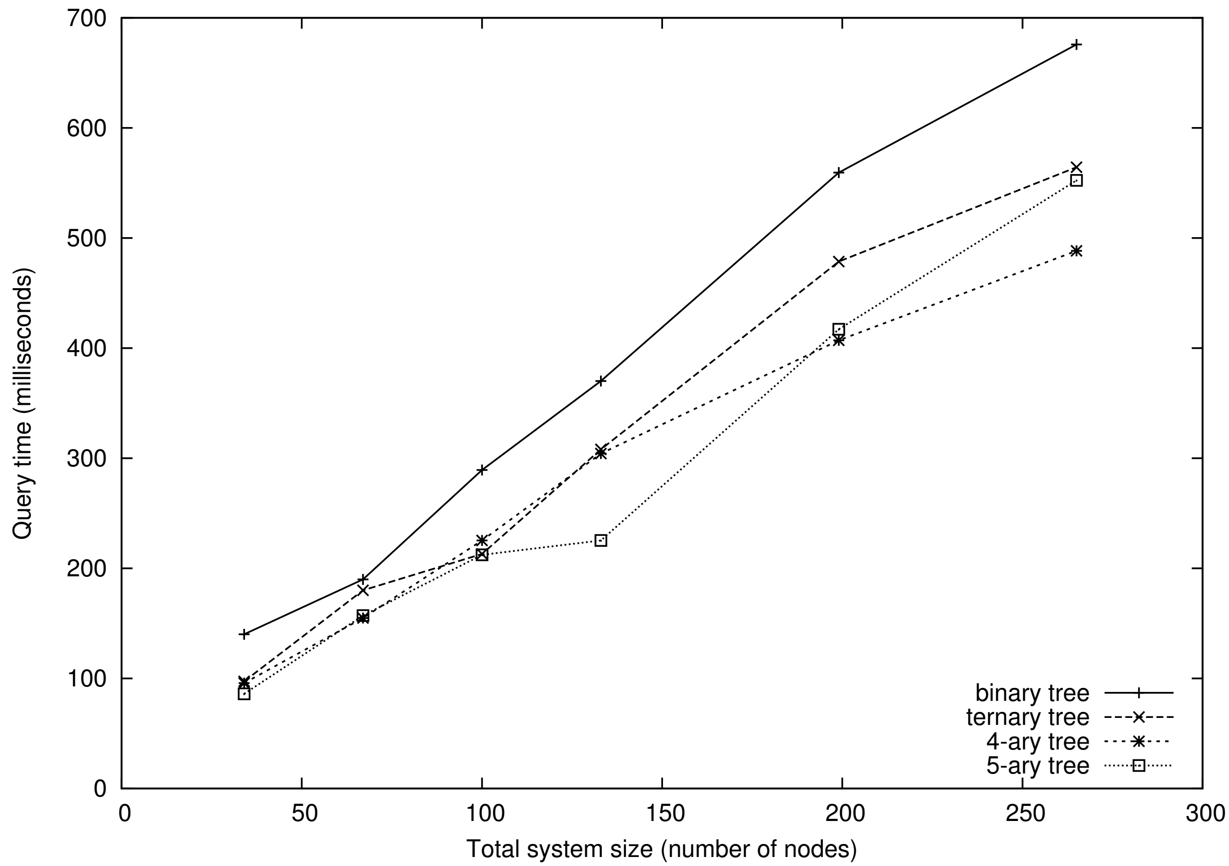
- Following results are of interest
  - Time to query various configurations of Fountain daemons
  - Time to recovery tree topology from single and multiple failures
  - Time to rebuild tree topology (worst case)
  - Quantify compute node overhead

Test Environment 3/3

- Why are these results important?
  - Original design goals were: fault tolerance, low overhead, good scalability
  - Scalability: time to query should scale somewhat linearly with number of nodes
  - Fault tolerance: recovery time should not depend on number of nodes

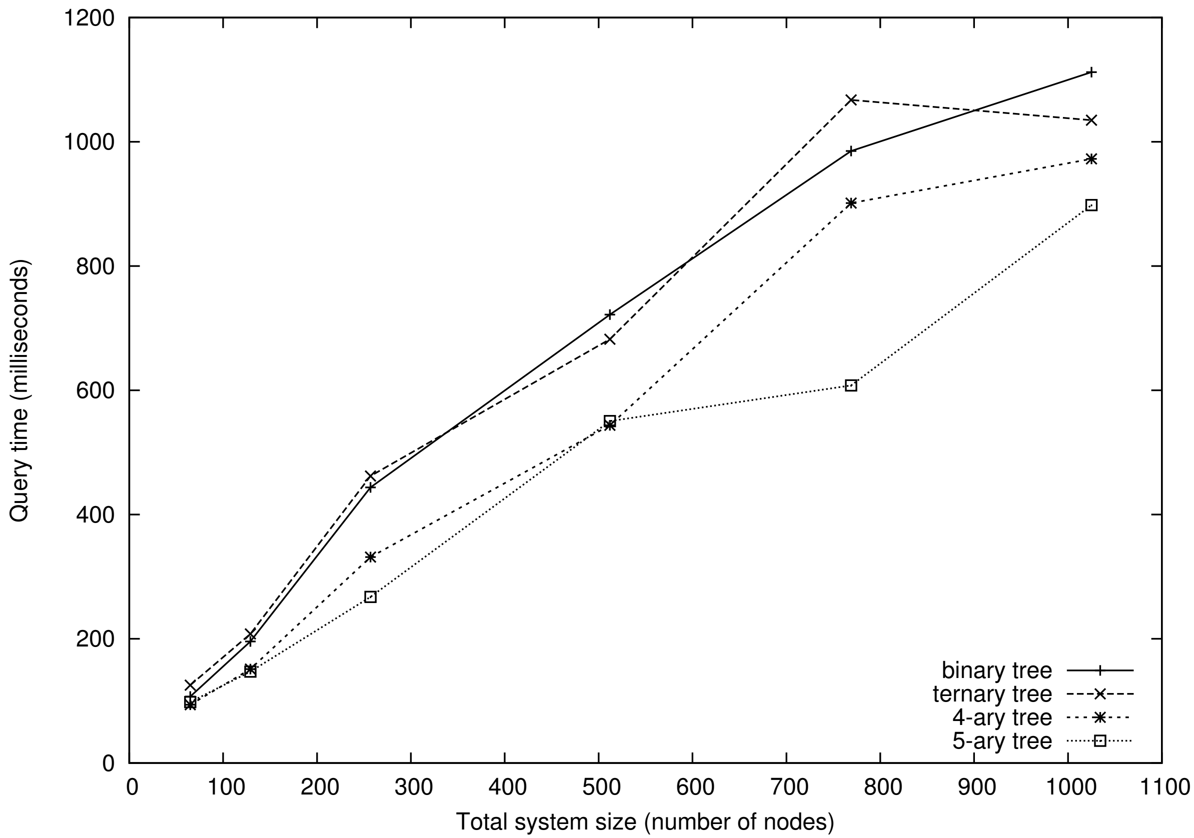
Node Query Results 1/2

Elapsed node query time (milliseconds) on 4pack
using 16 different tree configurations



Node Query Results 2/2

Elapsed node query time (milliseconds) on Scink
using 16 different tree configurations

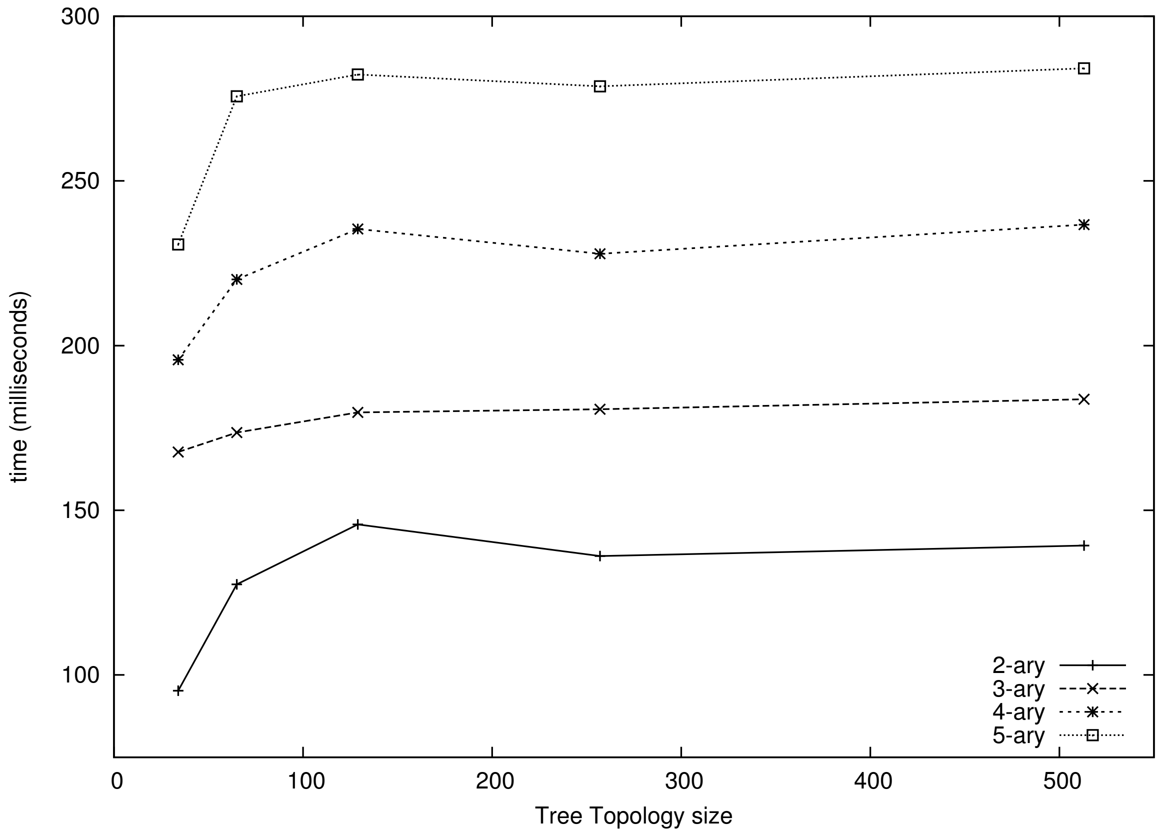


Tree Recovery 1/3

- Recovering from a single node failure
  - Measured as time when first node reports failure, until replacement node is connected
  - We expect tree topology with larger degree to require more time
  - Good scalability is important here, failure rates will rise as cluster sizes increase

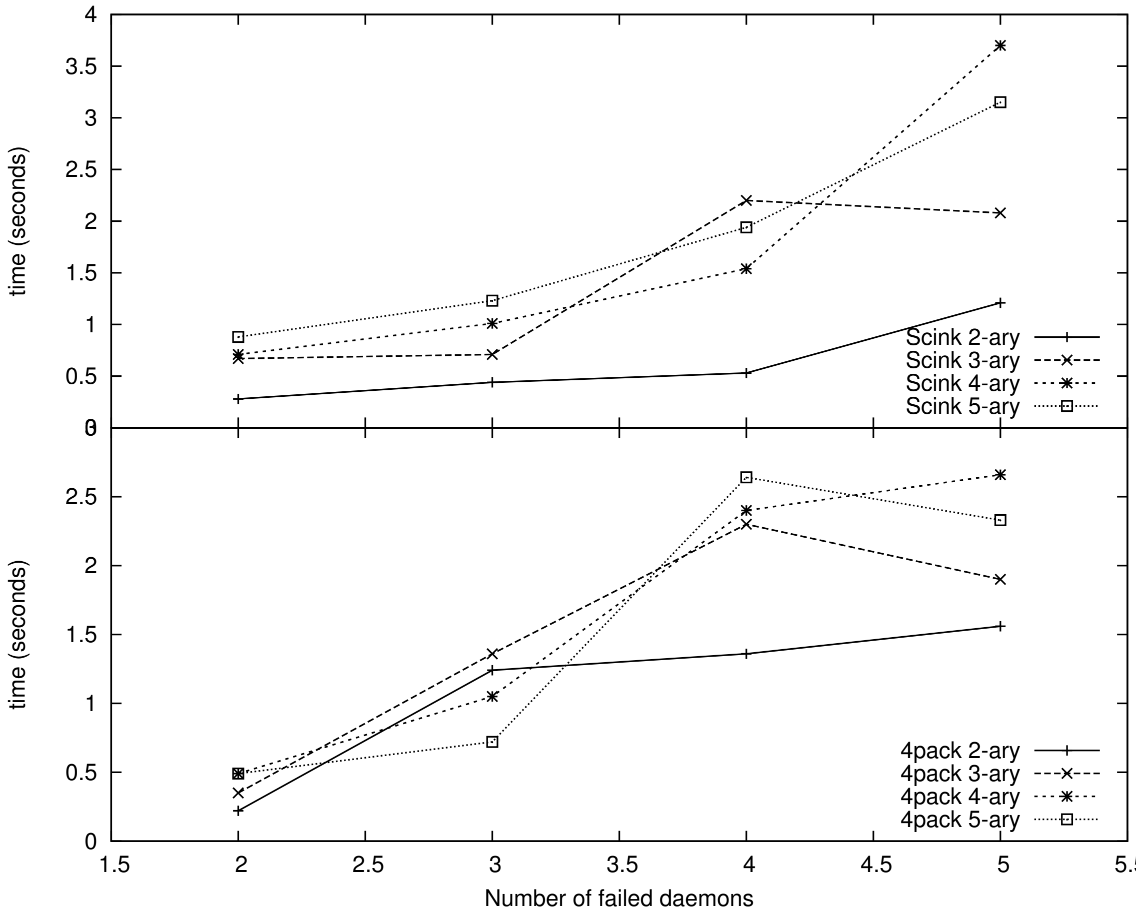
Tree Recovery 2/3

recovering from a single node failure
34 daemons on 4pack
65, 129, 257, and 513 daemons on Scink



Tree Recovery 3/3

recovering from multiple node failures on both 4pack and Scink

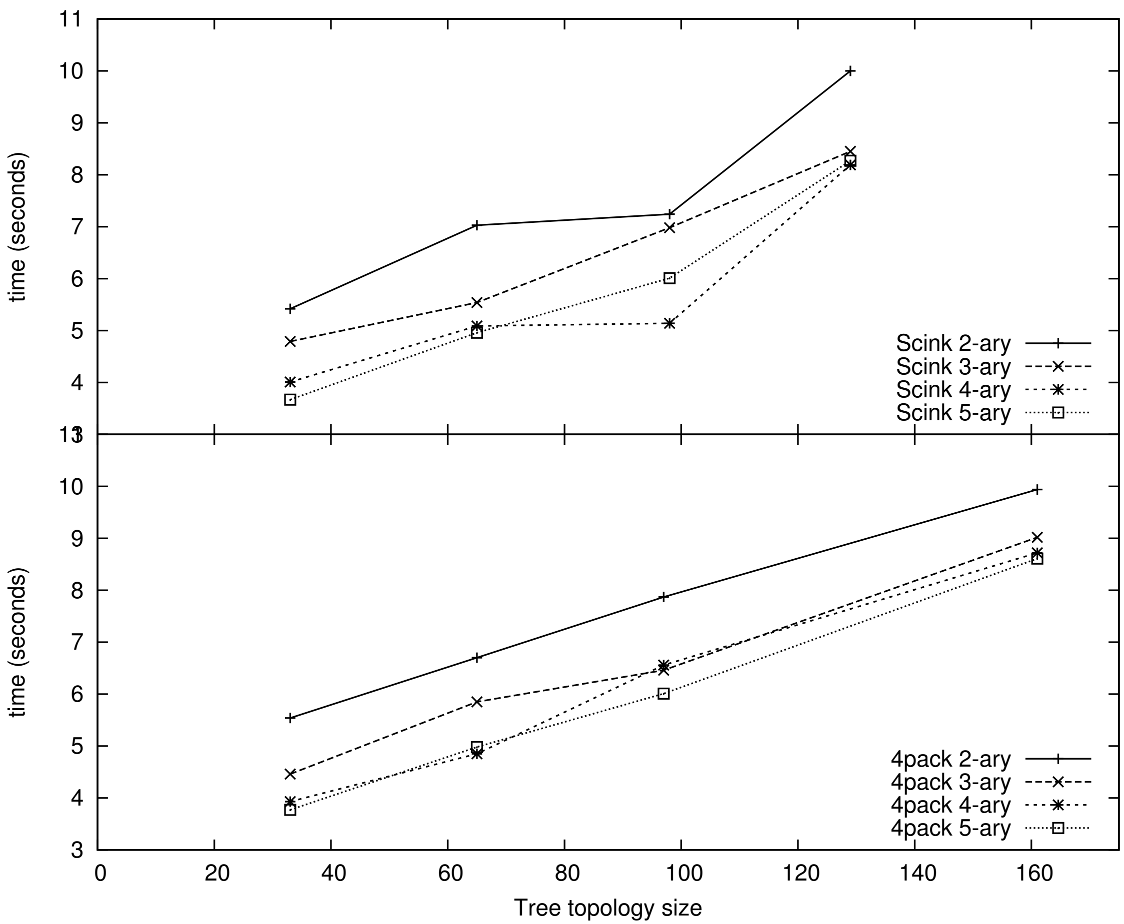


Tree Rebuilding 1/2

- This algorithm is a last ditch effort if recovery is not possible
- Requires master daemon to talk with each slave daemon in the system (EXPENSIVE)
- Performance numbers here are the result of a forced rebuild

Tree Rebuilding 2/2

rebuilding the tree topology
on both 4pack and Scink

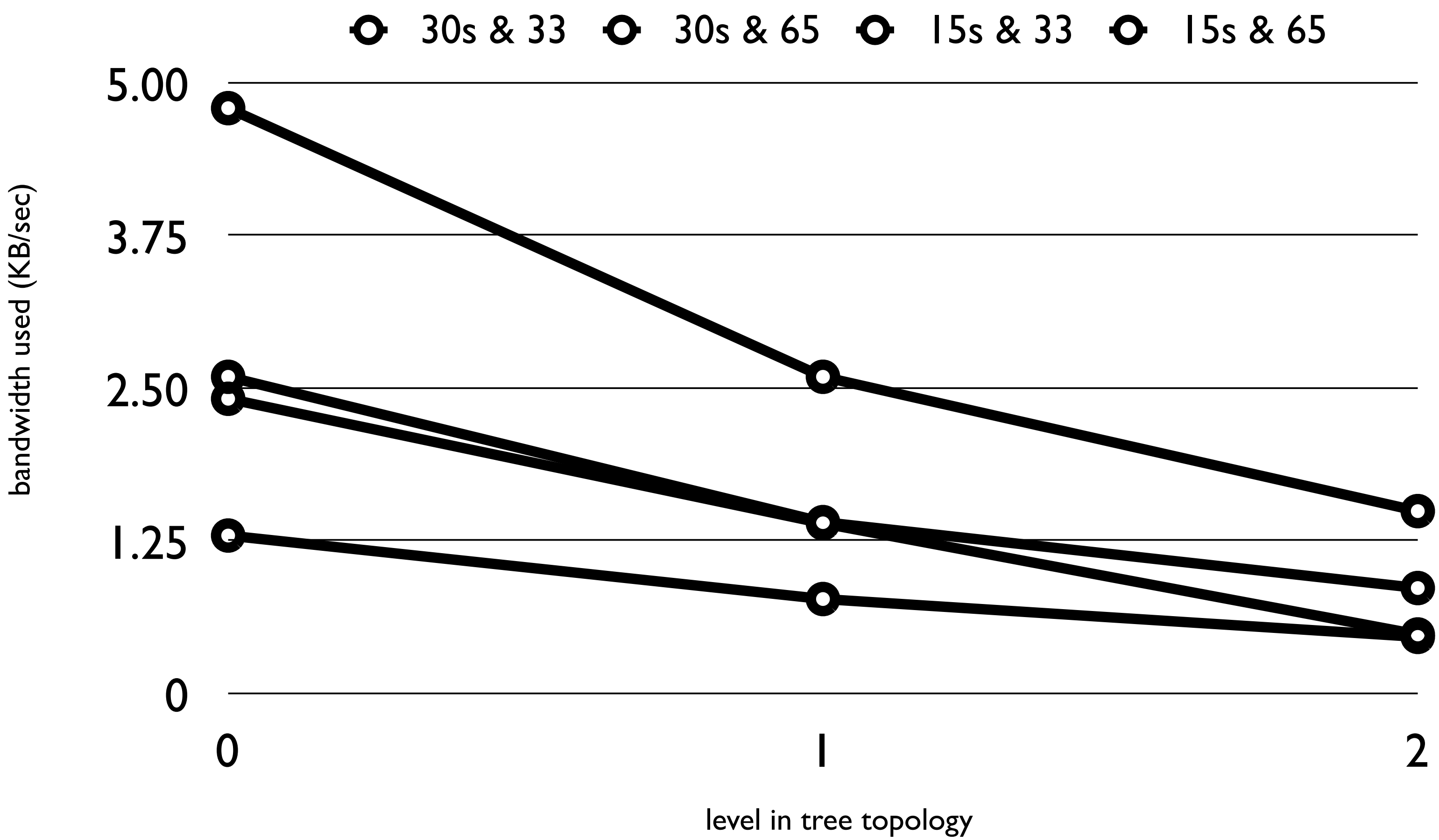


Node Overhead

- Design goal was low node overhead
- Compute nodes especially
- Can be quantified in terms of CPU usage and network bandwidth
- Several parameters can affect these numbers
  - Location in tree topology
  - Fountain server query interval
  - Size of tree topology

Node Overhead

(in terms of query interval and tree size)



Future Work

- Consider use of threads and non-blocking sockets
- Consider multiple master daemons and server processes for fault tolerance
- Find optimal tree topology degree based on cluster information
- Improve Goanna administrative GUI

Conclusion

- Fountain is a node monitor for the Scalable Systems Software project
- It utilizes a component based design to pull information from each node in the cluster
- Our major research contribution is the use of a rigid tree topology of persistent daemons
  - Promotes good scalability
  - Recovering from failures depends on the topology degree

Publications

- S. Miller and B. Bode. *The Node Monitoring Component of a Scalable Systems Software Environment.* In Proceedings, IEEE International Conference on Parallel and Distributed Systems. Minneapolis, MN, July 2006.

Acknowledgments

- Thank you for attending today
- Special thanks to SCL staff & colleagues
- This research project is supported by the United States Department of Energy
- The two clusters used to develop and test this research are supported by the DOE MICS office

Questions?